2. The DII COE

The concept of a Common Operating Environment (COE) as embodied in the DII COE is perhaps the most significant and useful technical byproduct of the Joint Service/Agency technical meetings that led to the successful GCCS development effort. It represents the culmination of several years of development amongst the services/agencies and it is interesting to note that the services/agencies independently arrived at similar conclusions. The DII COE encompasses architecture, standards, specifications, software reuse, shareable data, interoperability, and automated integration in a cohesive framework for systems development. Automated integration is described more fully in Chapter 3.

Yet the COE is one of the most widely misunderstood concepts. The COE is in reality very simple and straightforward, but powerful in its ability to tailor a system to meet individual site and operator requirements. Much of the confusion can be traced to uncertainty over what software is or is not in the COE, whether or not the COE and GCCS are synonymous terms (they are *not*), and the evolutionary development nature of the COE. The requirement to establish a migration strategy that preserves legacy systems and the near-term objective of replacing WWMCCS are responsible for a great deal of confusion, but the need to evolve the COE is driven by many factors and the architectural freedom to evolve is one of the strengths of the COE.

This chapter is devoted to explaining the DII COE in detail. Definition of a COE is principles-driven, not application-driven, so this chapter begins with a discussion of those principles. Selection of the actual components to populate the COE creates a COE *reference implementation*. This is important because the components which constitute a COE instantiation determine the specific mission domain that a COE can address (e.g., C4I for GCCS, logistics for GCSS, finance for ECPN), and how broadly defined the mission domain can be. Because the COE is structured so that only required components are loaded, a properly defined COE is suitable for a service-specific system (e.g., Navy JMCIS, Air Force BSD) or a joint system (e.g., GCSS, ECPN). Also, because the architecture is principles-driven, the DII COE is extensible to larger mission domains by expanding the selected set of software components. The COE is an open architecture whose principles apply equally well to Unix[1] and non-Unix platforms such as the PC. The DII COE contains a reference implementation for both Unix and NT platforms.

Subsection 2.1 discusses fundamental COE concepts and also describes what is meant by DII compliance and interoperability. As with any standard, compliance is required to avoid conflicts that prevent interoperability. Take careful note in reading subsection 2.1 that the discussion is relevant to any COE-based system since the principles apply much more broadly than to a single system such as GCCS or GCSS. Remaining subsections elaborate on software and hardware configurations selected for support by DISA and how the software is structured at a top level to limit site operators to only those functions they are authorized to access.

[1] Unix in this document is used in the sense of a vendor-proprietary implementation of the "traditional" Unix operating system. Although desirable, it is not necessary that vendors have received an X/Open UNIX 95 branding.

2.1 Fundamental COE Concepts

In COE-based systems, all software and data - except certain portions of the kernel (see subsection 2.1.2.1 below) such as the operating system and basic windowing software - are packaged in self-contained units called *segments*. This is true for COE infrastructure software and for mission-application software as well. Segments are the most basic building blocks from which a COE-based system can be built. Segments are defined in terms of the functionality they provide, not in terms of "modules," and may in fact consist of one or more CSCIs (Computer Software Configuration Items). They are defined as a collection of related functions as seen from the perspective of the end user, not the developer. The reason for defining segments in this way is that it is a more natural way of expressing and communicating what software features are to be included in, or excluded from, the system than by individual process, file name, or data table. For example, it is more natural to think of a system as containing a message processing segment than CSCIs called Icm, Ocm, and Pcm. It is more natural to the end user to think of a word processor segment than a software CSCI that opens a file, another CSCI that paginates a file, another CSCI that compresses a file, etc.

Those segments that are part of the COE are known as *COE-component segments*, or more precisely, as segments that further have the attribute of being contained within the COE. Segments that are built on top of the COE to provide capabilities specific to a particular mission domain are *mission-application segments*. The principles which govern how segments are loaded, removed, or interact with one another are the same for all segments, but COE-component segments are treated more strictly because they are the foundation on which the entire system rests. A later chapter further refines the segment concept to distinguish between data segments, software segments, patches, etc. but the point here is that segments are a technique for packing system components.

Each segment in the system contains a directory with a collection of data files that "self-describe" the segment to the rest of the COE. The directory that contains these files is called the *segment descriptor directory* and the files themselves are called *segment descriptors*. The process of decomposing a component into individual packages and creating the required segment descriptors is called *segmentation*.

Packaging a system in terms of segments along with the strict rules which govern the COE and runtime environment provide several immediate benefits:

- *Segment developers are decoupled and isolated from one another.* Segments are self-contained within an assigned directory. Developers have maximum freedom within the assigned segment directory, but minimum freedom outside it. This allows multiple developers to work in parallel with seamless integration after development.

- *Extensions to the COE are coordinated through automated software tools.* It is not possible to create a single configuration of the COE that meets all possible mission-application or site-unique requirements. However, the COE tools make it possible to extend a base COE in a carefully controlled way to ensure compatibility and identify segment dependencies and conflicts.

- *Compliance verification and installation can be automated.* Standards without automated validation are difficult to use in practice, especially in a program where the system is large and there is a need to coordinate activities from several different contractors, program sponsors, services, and agencies. The COE approach to validation is closely related to software installation so that automation of one directly leads to automation techniques for the other.

- *Mission-application segments are isolated from the COE.* System integration problems are frequently a result of an undisciplined interaction between software components or because of tight coupling between components. The COE controls interaction through APIs and isolates mission applications from the COE-component segments so that failure of one mission-application segment is less likely to affect another or affect the stability of the COE foundation itself.

- *Segments created by one developer for one system can be readily reused by another developer for another system*. That is, the DII COE is an effective strategy that includes not just software reuse, but also ensures that a reused segment fits seamlessly into the new system.

- *Integration is simplified and the original developers resolve most integration problems before the segment is ever submitted*. The segment descriptors "self-describe" the segment so that all pertinent information required to integrate the segment into a system is contained in a standard, known location. The tools that validate conformance to the COE detect a large percentage of traditional integration difficulties. Moreover, the process of integration is largely automated as a byproduct of the installation tools themselves. By its very nature, the DII COE process pushes integration responsibilities further down to the original developer than is done with more traditional approaches.

- *Configuration Management is simplified*. One way that the COE process simplifies configuration management is by using segment descriptors that allow dependencies on, or conflicts with, other segments to be expressed. It then becomes possible to express the requirement for a top-level functional capability (e.g., a tool for editing an Air Tasking Order) and then recursively traverse a dependency tree to identify all required segments for the desired capability.

These benefits apply equally well to Unix and NT environments and are in fact not dependent upon the underlying operating system.

The DII COE is a *superset* of capabilities. It contains far more functionality than would ever be installed on a single workstation or even at a specific operational site. Thus, it is important to note and understand that just because a segment is part of the COE, it is not necessarily always present or required. Considerable flexibility is offered to customize the environment so that only the segments required to meet a specific mission-application need are present at runtime. This approach allows minimization of hardware resources required to support a COE-based system.

To illustrate the point, consider an example. The COE contains a service for displaying maps. However, some C4I operators in command centers only need to read and review message traffic and do not need or want to view a tactical display. Logistics operators using GCSS do not need to see the tactical picture at all and may only desire to see a map when planning transportation routes. For such operators it is not necessary at runtime to have the extra memory and performance overhead of the segments that generate cartographic displays.

Understanding the concept of a segment is fundamental to understanding and using the DII COE. It is, however, only the starting point. Given the background on how COE-based systems are packaged, it is now time to understand the internal structure of the DII COE.

2.1.1 COE Taxonomy

Segments that comprise the COE can be categorized in several ways. The original GCCS COE was subdivided into 19 functional areas and was organized largely by technologies employed such as network, database, and MCG&I (Mapping, Charting, Geodesy, and Imaging). Working groups were established for each of the 19 functional areas to consolidate operational requirements from each of the services/agencies and to evaluate and recommend candidate modules as core components. This taxonomy was initially successful and led to several early successes. However, the large number of working groups defined by this taxonomy quickly became unwieldy and communication within and between working groups became infeasible.

The DISA COE Design Working Group revisited the COE taxonomy as part of the effort to expand the GCCS COE into a DII COE. The present taxonomy consists of two layers: Infrastructure Services and Common Support Applications[2]. These two layers are described in more detail in the *Architectural Design Document for the Defense Information Infrastructure (DII) Common Operating Environment (COE)*, and summarized in Figure 2-1. While encompassing the same functionality as the original 19 functional areas, this taxonomy approaches the problem from an architectural perspective rather than functional, and greatly reduces the communications burden in and between working groups. Figure 2-1 will be updated to include other functional areas as appropriate as the COE is extended to other mission domains. It has been updated since the *Architectural Design Document* to extend it for logistics support and to include a Web Server. This server is provided to allow access to COE-based applications from a Web browser. A later chapter in the *I&RTS* describes the COE Web in more detail.

[2] The concepts of a *COE kernel* and *SHADE* are presented later in this section. Both of these concepts should be viewed as subsets of the Infrastructure Services and Common Support Applications layers. The COE kernel is a limited subset of the Infrastructure Services that is required on every platform regardless of how it will be used. SHADE is a subset of Infrastructure Services and Common Support Applications that deals with database issues. It is frequently useful to discuss the kernel and SHADE as separate entities, but their functionality is fully contained within the two layer taxonomy discussed in this section.

The difference between Infrastructure Services and Common Support Applications is the difference between *data* and *information* (i.e., processed data). It is the difference between *exchanging* data and *sharing* data. Infrastructure Services provide low-level tools for data exchange. These services provide the architectural framework for managing and distributing the flow of data throughout the system. Example services include TCP/IP and UDP protocols, DCE, and CORBA. The achievement of effective data sharing requires use of all the COE services, especially those provided by SHADE (Shared Data Environment). Subsection 2.1.2.5 below describes SHADE in more detail.

Common Support Applications, on the other hand, provide the architectural framework for managing and disseminating information flow throughout the system, and for sharing information among applications. This level contains facilities for processing and displaying common data formats and for information integration and visualization. Services in this layer tend to be mission-domain specific. Examples include generation and dissemination of mission-relevant alerts, and word processing support.

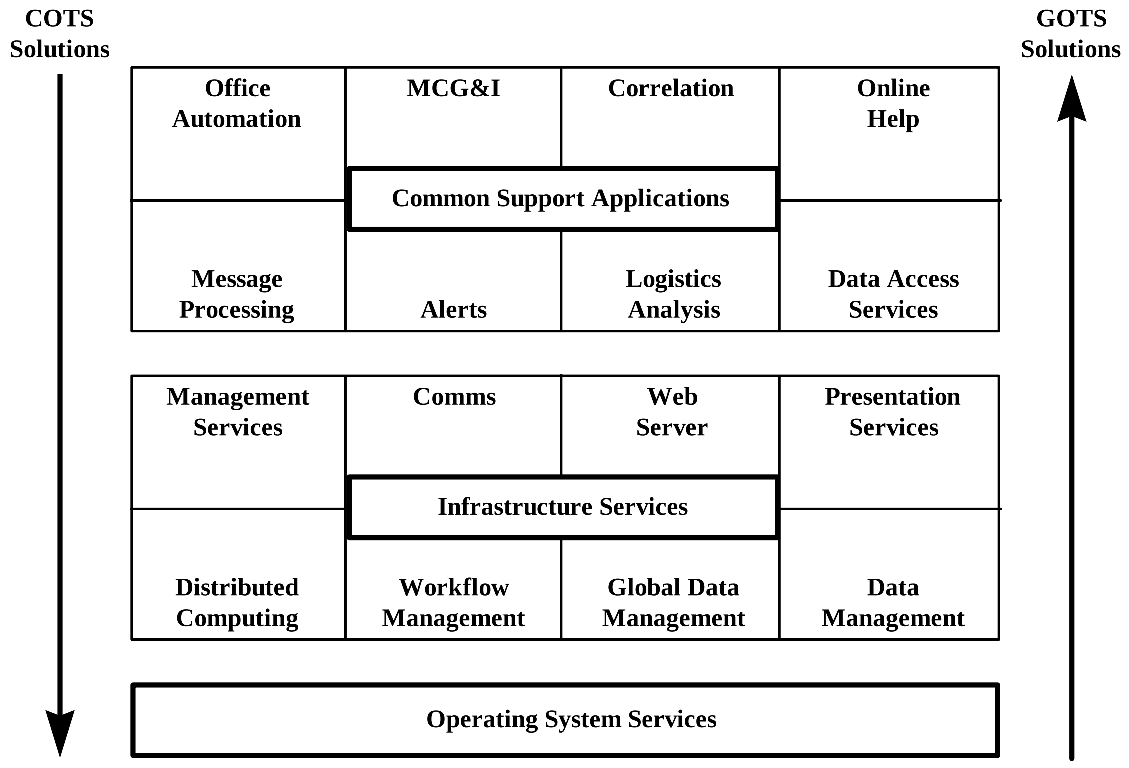

Figure 2-1: DII COE Services

Figure 2-1 also shows that there is a relationship between the service provided and whether it is typically provided by a COTS product or a GOTS product. The DII COE uses COTS whenever possible, in keeping with DOD directives. Infrastructure Services are normally provided by COTS solutions because they are closely tied to underlying vendor products such as the operating system. Common Support Applications, because the services they provide are closely related to mission applications, tend to be provided by GOTS solutions. In some cases, especially in the Office Automation area, services may include COTS solutions.

Selection of software modules that fulfill these COE component responsibilities is an ongoing task as is the evolutionary nature of the DII COE. Changes are made to further populate the COE, to optimize selected components, or to extend the COE to meet requirements from other mission domains. Even though the process is evolutionary, the COE preserves backwards compatibility so that mission applications are not abandoned just because there is an update of the COE. Refer to the appropriate API, User's Guide, and system release documents for detailed information on the components currently selected for the COE.

2.1.2 COE Architecture

Figure 2-2 is a simplified diagram that illustrates the various levels within the DII COE and the relationship between the COE, component segments, mission-application segments, and SHADE. As can be seen, the COE encompasses APIs, GOTS and COTS software, the operating system, windowing software, standards (*TAFIM*), and specifications (*Style Guide*, *I&RTS*, etc.). Physical databases (MIDS IDB, JOPES database, etc.) are also considered to be part of the COE[3], including the software (such as the RDBMS), which accesses and manages the data. SHADE is an integral part of the DII COE, and encompasses databases and related software as noted in the diagram. SHADE and each of the layers are described in more detail below. A Developer’s Toolkit is also provided in the COE as shown in Figure 2-2.

Figure 2-2 is a generic diagram intended only to show relationships. The labeled boxes in the figure are not intended to be exhaustive, but are representative services because (a) otherwise the diagram would be needlessly complicated, and (b) the COE is evolving to include other segments to support new mission domains. The services shown are representative, but the structure and principles discussed are the same across all mission domains.

To use a hardware analogy, the COE is a collection of building blocks that form a software “backplane.” Segments “plug” into the COE just as circuit cards plug into a hardware backplane. The blocks containing the operating system and windowing environment are akin to a power supply because they contain the software which “powers” the rest of the system. The segments labeled as COE-component segments are equivalent to already built boards such as CPU or memory cards. Some of them are required (e.g., CPU) while others are optional (e.g., a specialized communications interface card) depending upon how the system being built will be used. The blocks in Figure 2-2 labeled as mission application areas are composed of one or more mission-

[3] In previous COE releases, physical databases were considered outside the scope of the COE although the database software was inside the COE. The programmatic decision to temporarily exclude physical databases was made in order to concentrate on the services that would more directly support mission applications. Databases are now included in the DII COE as part of the SHADE.

application segments. These segments are equivalent to adding custom circuit cards to the backplane to make the system suitable for one purpose or another.

The API layer shown in Figure 2-2 defines how other segments may connect to the backplane and utilize the “power supply” or other “circuit cards.” This is analogous to a hardware schematic diagram that indicates how to build a circuit card that will properly plug into the backplane. The figure also implies that APIs are the only avenue for accessing services provided by the COE. This is true for all COE software and all layers, including COTS software. However, the COE does not create an additional layer on top of the COTS software. These components may be accessed directly using vendor-supplied APIs for these commercial products as long as such usage does not circumvent the intended COE architecture. For example, the COE includes a POSIX-compliant operating system. Some vendors provide non-POSIX compliant extensions to the operating system services. Use of such extensions, even though they are readily available through vendor-supplied APIs, is not allowed because such usage violates the intended COE architecture.

This hardware analogy can be extended to the SHADE portion of the COE, but with some significant distinctions. Within this conceptual model, the DBMS functions as the COE’s disk controller and disk drives. The applications’ databases can be equated to directories or partitions on the drives accessed through the DBMS “disk controller.” Data objects belonging to each database then can be considered as files within those “directories.”

This analogy is critical to understanding the modularity limitations for databases within the COE. One can replace most peripherals or circuit cards without any side effects just as one can replace mission applications without losing information. However one cannot put in a larger disk drive, or change from one type of controller to another, without losing the data on the disk. While upgrading mission applications is like swapping circuit cards, upgrading databases is like rebuilding a disk or directory structure. Instead of replacing a component, one must save and then restore the files on the disk. Proper design of COE/SHADE databases must provide the ability to perform field upgrades without the loss of any data.

COE/SHADE databases are divided among segments as are mission applications, but with a different focus. Mission applications are segmented based on their functionality. Databases are segmented functionally by the subject areas of the mission applications they support. Mission applications are functional modules; databases are information modules.

The precise configuration of COTS products used in the COE is placed under strict configuration control. This is necessary because configurable items such as the amount of shared memory or swap space must be known and carefully controlled in order for other components in the COE to operate properly. For this reason, COTS products are assigned a version number in addition to the vendor-supplied version number so as to be able to track and manage configuration changes. Databases are also assigned version numbers because their configurations must be controlled since the data content may change from release to release, or the database schema may change.

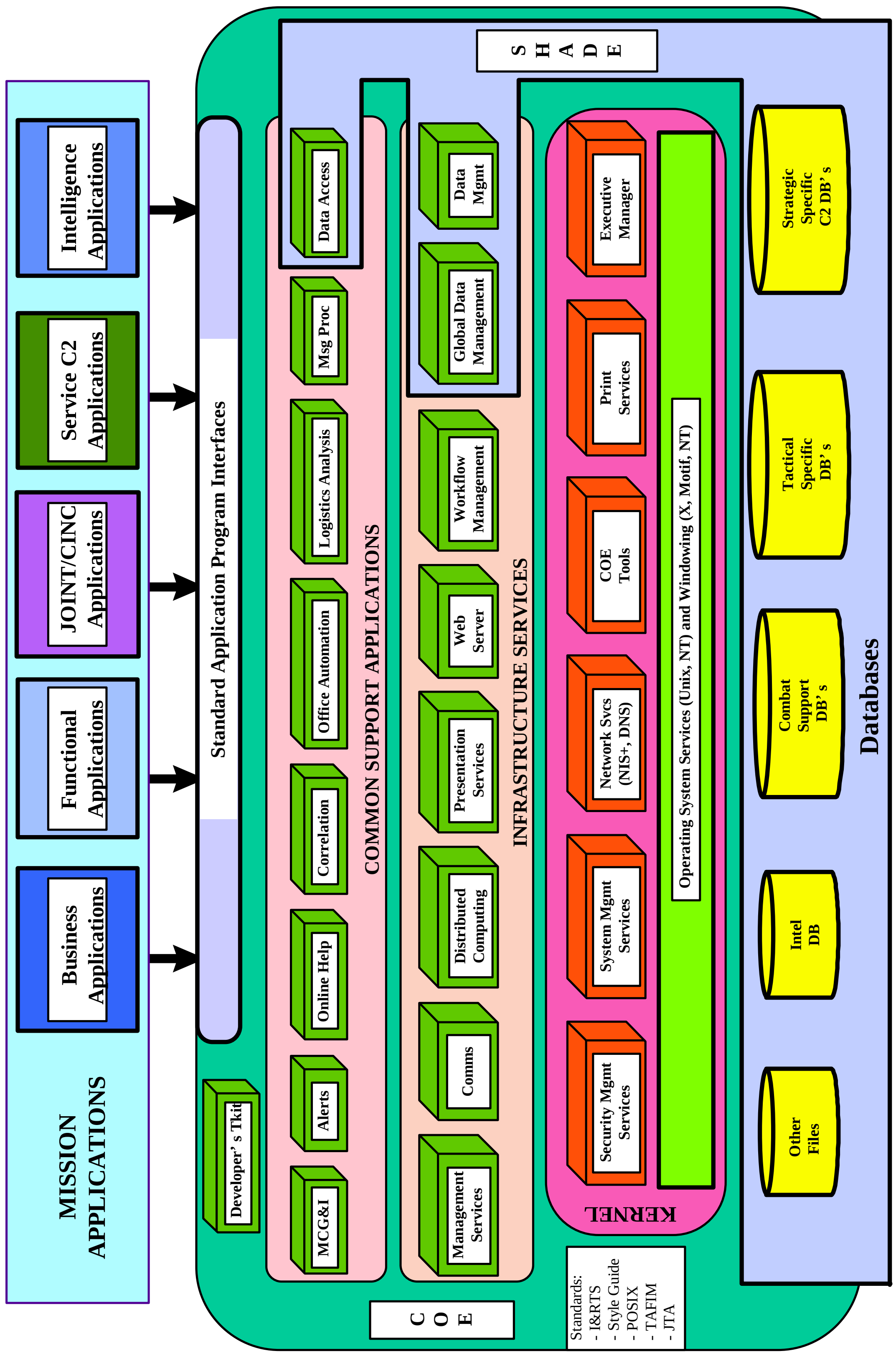

Figure 2-2: DII COE Architecture

A fundamental principle throughout the COE is that segments are not allowed to directly modify any resource “owned” by another segment. This includes files, directories, modifications to the operating system, and modification to windowing environment resources. Instead, the COE provides tools through which a segment can request extensions to the base environment. The importance of this principle cannot be overemphasized because environmental interactions between software components are a primary reason for difficulties at integration time. By providing software tools that arbitrate requests to extend the environment, integration can be largely automated and potential problem areas can be automatically identified.

For example, the COE predefines a set of ports in the Unix `/etc/services` file. Some segments may need to add their own port definitions, but this will create conflicts if the port definitions are the same as those defined by the COE or another segment. To identify and prevent such conflicts, segments issue a request to the COE (see Chapter 5 for how this is done) to add their port definitions. This process is called *environment extension* because a segment is modifying the predefined environment by extension, not through replacement or deletion.

COE-component segments shown in Figure 2-2 are typically designed to be servers, although some are provided as libraries to be linked with an application segment. Note that in practice such segments will often operate in both a client and server mode. For example, a track management segment is a server for clients that need to retrieve the current latitude/longitude location of a platform. But the track manager itself is a client to a communications server that initially receives track-related reports from sensors or other sources. Refer to the *Architectural Design Document for the Defense Information Infrastructure (DII) Common Operating Environment (COE)* document for more detailed discussion of how COE-component segments are designed and interact. For purposes of the present discussion, it is sufficient to view COE segments as servers that are accessible through APIs.

2.1.2.1 COE Kernel

The COE will normally make available a large number of segments, not all of which are required for every application. The *kernel COE* is the minimal set of software required on every workstation regardless of how the workstation will be used. The kernel COE components are shown in Figure 2-2 and include the Operating System and Windowing Services and a collection of other services that properly belong in the Infrastructure Services Layer.

A kernel COE will always contain the operating system and windowing environment, but it will normally include six other features:

1. a basic System Administration function,
2. a basic Security Administration function,
3. an Executive Manager function (e.g., a desktop GUI such as Windows NT or CDE),
4. a template for creating privileged operator login accounts,
5. a template for creating non-privileged operator login accounts, and
6. COE tools for segment installation.

The System Administration segment is required because it contains the software necessary to perform basic system administration tasks such as user account and profile management. The Security Administration segment is required because the security administrator uses it to enforce system security policy. The operating system and other COE components provide security policy enforcement. Segments loaded later may provide additional system and security administration capabilities, but the minimum capabilities for security enforcement and security administration are in the kernel.

The Executive Manager component of the kernel is required because it is the interface through which an operator issues commands to the system. The Executive Manager is an icon-and-menu-driven desktop interface, not a command-line interface. The templates included in the kernel COE are used to define the basic runtime environment context that an operator inherits when he logs in (which processes to run in the background, which environment variables are defined, etc.). The COE tools within the kernel allow other segments to be installed and enforce critical COE principles. The kernel COE assures that every workstation in the system operates and behaves in a consistent manner and that every workstation begins with the same environment.

From an installation sequence perspective, it is necessary to define a subset of the kernel COE called the *bootstrap COE*. The reason is that segments are installed through a special COE program, the segment installation tool. The segment installation tool is accessed as a System Administration function. However, the segment installation tool itself must be installed before it can be used to install segments. Moreover, COTS software is not usually in segment format. How then are the segment installation tool and at least a minimum operating system installed to permit loading the kernel COE? This is typically done by loading the operating system and windowing environment, as directed by the hardware vendor, and then the COE segment installation software is loaded. Once the COE is thus “bootstrapped,” it is possible to load the remaining components of the kernel COE and any additional segments.

2.1.2.2 Infrastructure Support Services

Within the Infrastructure Services layer, Management Services include network, system, and security administration. Communications Services provide facilities for receiving data external to the system and for sending data out of the system. Distributed Computing Services provide the infrastructure necessary to achieve true distributed processing in a client/server environment. Presentation Services are responsible for direct interaction with the human whether that be through windows, icons, menus, or multimedia. Data Management Services include relational database management as well as file management in a distributed environment. Workflow and Global Data Management Services are oriented towards managing logistics data (e.g., parts inventory, work in process). Note that Data Management Services and Global Data Management Services are part of SHADE.

2.1.2.3 Common Support Applications

Infrastructure Services are largely independent of any particular application. Common Support Applications tend to be much more specific to a particular mission domain. The Alerts Service is responsible for routing and managing alert messages throughout the system whether the alert is an "out of paper" message to a systems administrator or an "incoming missile" alert to a watch operator. The Correlation Service is responsible for maintaining a consistent view of the battlespace by correlating information from sensors or other sources that indicate the disposition of platforms of interest. MCG&I Services handle display of National Imagery and Mapping Agency (NIMA) maps or other products, and imagery received from various sources. Message Processing Services handle parsing and distribution of military-format messages. Office Automation Services handle word processing, spreadsheet, briefing support, electronic mail, World-Wide-Web browsers, and other related functions. (Browsers are in the Common Support Applications layer, but Web Servers fall within the Infrastructure Services layer.) Logistics Analysis contains common functions, such as Pert charts, for analyzing and displaying logistics-related information. Online Help Services provide applications with a uniform technique for displaying context-sensitive help. Finally, Data Access Services are part of SHADE and provide applications with common data-access methods procedures, and tools.

2.1.2.4 COE Developer Toolkits

Since the COE is not a system but a foundation on which systems are built, the COE contains a collection of developer toolkits to assist the developer in creating mission-application software. This is illustrated in Figure 2-2 in the block labeled Developer's Toolkit. However, the toolkits are required only during software development, not during runtime at an operational site. Therefore, developer toolkits are shown as part of the COE, but outside the Infrastructure Services and Common Support Applications layers. They are obtained from DISA separate from an actual installable system.

The COE developer toolkits contain libraries of APIs and a collection of tools to assist in the segmentation process. An overview of the software development process is presented in the next chapter. Appendix C provides an overview of the COE developer tools. Refer to the appropriate *DII COE Programmer's Guides* for detailed information on the APIs and segmentation tools.

2.1.2.5 SHADE

SHADE is an important addition to this version of the *I&RTS*. Its purpose is to provide the data "missing piece" for the DII COE. The present subsection provides an overview of SHADE and describes how it fits into the overall DII COE. A later chapter will cover SHADE and database topics in much more technical detail and depth.

Present systems are not truly interoperable because of inconsistency in algorithms, but also because data management across systems and operational sites has led to data redundancy and inconsistencies. Moreover, even when data is consistent across systems, it is not presently structured so as to be shareable. The SHADE approach is to provide the architectural structure to solve the data sharing problem that in turn guarantees data consistency, eliminates redundancy[4], and promotes true data interoperability and sharing. The SHADE goal is to allow any authorized user from any authorized workstation to locate, access, and integrate shared and synchronized data. This is in keeping with the DISA vision of an integrated global environment that allows warriors to perform "Any Mission, Any Time, Any Where."

SHADE is both a strategy for data sharing and the mechanisms to achieve it. SHADE is an integral part of the DII COE, but it must also bridge the gap between COE-based systems and legacy non-COE systems because it must provide mechanisms for accessing large databases that are still on legacy mainframes. SHADE provides COE-component segments in both the Infrastructure Services and Common Support Applications layers to accomplish this task. SHADE includes the required data-access architectures, data sharing methodology, reusable software and data components, and guidelines and standards for the development and migration of systems that meet the user's requirements for timely, accurate, and reliable data.

The SHADE components of Figure 2-2 are expanded upon in Figure 2-3 to show the architecture from a data management perspective. From a process point of view (top part of the diagram), SHADE includes tools for validating database segments and a repository for data reuse. Metadata Management is at the top layer between the mission applications and data-access methods. This layer is among the more challenging aspects of SHADE because it requires standardization across the joint community. The Shared Data Access layer provides services for locating and retrieving the desired data. This layer also manages data replication and distribution to ensure that all databases are kept closely synchronized. Data security is also provided in this layer.

The Physical Data Management layer is provided by commercial products and is initially organized as relational databases. (Migration to include other database management technologies such as object-oriented or object relational will be achieved as requirements emerge and technology matures.) SHADE physical data management services may also include document retrieval, image management, CAD/CAM drawings, or other specialized

[4] Databases are often deliberately replicated in actual practice for performance reasons. The term "redundant data" is used when the same data is captured by different systems and stored in different databases. For example, the friend/foe status of a particular country might be entered into two systems where each system must maintain and keep the data current. By contrast, when intentional replication is used, the friend/foe status is captured and maintained by one system and provided to another for its use. SHADE may not rule out multiple copies of the same data but it does manage the duplication to ensure that all databases are kept synchronized. Present systems often do not employ effective mechanisms for data replication, leading directly to significant interoperability problems. SHADE does eliminate redundancy *between* systems because, for performance reasons, it replicates and manages duplication *across* systems to ensure data consistency.

storage and retrieval technologies where appropriate. The databases may be distributed across the network, and may in fact be distributed among geographical sites.

Figure 2-3 shows three types of database segments according to their scope and how they are shared. The three types are Unique, Shared, and Universal. *Unique* database segments are those which are typically used by only one application or are under the configuration control of the segment sponsor. Unique data may be shared between applications, but the usage is restricted to a single mission domain. An example of a Unique database segment is a configuration table that an application reads at initialization time. Such a table would not normally be used by other applications. This example also demonstrates that Unique database segments may frequently be represented by a flat file or similar structure rather than a true database.

Shared database segments support the information requirements of multiple applications or across multiple database segments. Shared database segments are typically mission-or-functionally-oriented, and are generally specific to a limited number of mission domains. Because they affect multiple applications that will likely span services or functional areas, Shared database segments must be under joint configuration control. An example of such a database segment is a database of logistics drawings for military hardware. Such data spans multiple services, it is used for different purposes (e.g., ordering, inventory control, maintenance) and hence spans multiple applications, but it is generally limited in scope to the logistics community. Another example is a segment containing invoice information that is required by both the finance and procurement communities.

Universal database segments represent the other extreme of “shareability.” Universal database segments reflect a need for identical data in diverse areas, are used by many applications, and span multiple mission domains. Universal database segments usually have no dependency on any other segment (except the DBMS segment) and frequently consist of a small number of tables and elements. A common type is reference or lookup tables. An example is a database of country-code abbreviations. A larger example would be the equivalent of “Jane’s Data” with data specifications on weapons, aircraft, ships, and communications systems. Universal database segments are under stricter configuration control and require DISA and DOD Data Administration coordination.

The three database segment types are listed in increasing order of scope and “shareability.” That is, Unique is limited in scope and therefore unlikely to be shared by many applications, while Universal is very broad in scope and *must* be shared across applications in order to promote true interoperability. There is no physical difference in the database segments, but the level of configuration management increases due to the wider impact changes would have on operational systems that use the database segments.

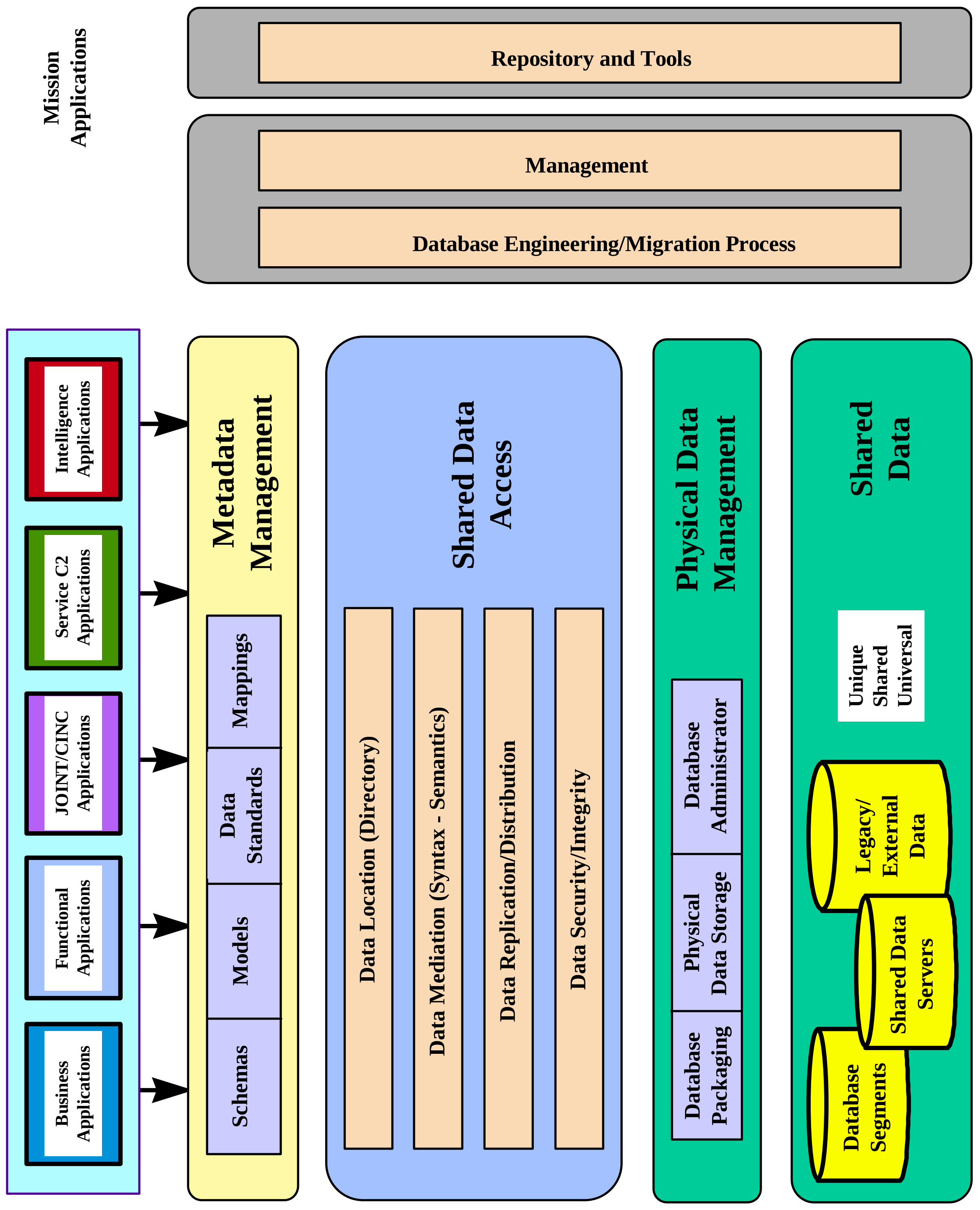

Figure 2-3: SHADE Data Architecture

2.1.2.6 COE "Plug and Play"

The DII COE is structured as a "plug and play" architecture. The key to the "plug and play" design is conformance to the COE through the rules detailed in this document and through using only the published APIs for accessing COE services. There is considerable danger in using unpublished, "private" APIs, or APIs from legacy systems, because there is no guarantee that interfaces used in this fashion will remain the same or even exist in subsequent releases. This is also generally true of COTS products and the risks are the same.

Discussion of the COE as a "plug and play" architecture is not intended to trivialize the effort that may be required to develop and integrate a segment into the COE. Migration of existing legacy systems to the COE is conceptually straightforward but may require considerable effort due to the requirement to switch to a different set of building blocks. That is, the effort may not be so much in adjusting to a new architectural concept but in adjusting code to use a different set of APIs. The "plug and play" paradigm is a good conceptual model because it clearly conveys the goal and the simplicity that most segment developers will encounter.

2.1.3 COE Configuration Definitions

A COE-based system will consist of a large number of segments. It is neither desirable nor feasible to install all segments on all platforms. Some segments need to be installed on one workstation but not another because of the role that the workstation will play in the overall system. For example, systems will often dedicate one or more workstations with large-capacity disk drives to be configured as database servers. Workstations that operators use, client workstations, will not have large enough drives to handle the database storage requirements. Therefore, the database server software should be loaded on the database server but not the client workstation. The COE kernel is required on every workstation, but additional segments are dependent upon how the workstation will be used.

The COE includes the ability to create configuration definitions that define which segments are to be loaded on which workstations. A *configuration definition*[5] is a hierarchy that defines collections of segments that are grouped together for installation convenience. For example, it is more convenient for an installer to indicate that a workstation is to act as a database server (a configuration definition) or used as an intelligence analyst workstation (another configuration definition) than to manually and individually select all of the segments that need to be installed. The COE is designed so that a site may install predefined configuration definitions or can customize the installation to suit site-specific requirements.

[5] The term *Configuration Definition* replaces the term *variant* in previous *I&RTS* releases. The concepts are exactly the same except that variant has the negative connotation of implying a "deviation." Further, the Configuration Definition concept is more refined in this *I&RTS* version in its decomposition into folders, configurations, and bundles.

A configuration definition is organized into folders, configurations, and bundles. Figure 2-4 uses an example from the GCCS system to show the relationship between each of these terms, and to illustrate the flexibility in predefining and managing software installations. The example shows how the GCCS system *could* be organized into configuration definitions, but not how GCCS *must* be organized. The example is not intended to convey that workstations must be dedicated to a single, specific function. As long as there are no segment conflicts, a workstation may be configured to support multiple missions and thus achieve the goal of "any workstation for any function." The example is intended only as an aid to understanding how configuration definitions may be constructed.

The objective of the example shown in Figure 2-4 is to install identical database servers in Intelligence centers at two GCCS sites: a CJTF and a CINCHQ. In this simplified example, both Intelligence centers use imagery applications, but the Intel center at the CINCHQ has access to hardware for capturing images, while the one at the CJTF does not.

For simplicity, the database server is to consist of 4 segments: a segment for creating database backups (`Bkup`), an ad hoc query application (`AdHocQ`), a presentation package (`Forms`), and a patch (`Patch1`). The imagery software is to consist of an application for creating briefs (`Brief`), an application for capturing images (`Capture`), and an application for converting images from one format to another (`Convert`).

A *configuration definition file* is a file that describes the hierarchy and relationships among folders, configurations, bundles, and segments that comprise a distribution. A *distribution*[6] is the physical media used to install DII-compliant segments (e.g., DAT tape, 8mm tape, CD-ROM). A single distribution may span multiple media of the same type (e.g., several DAT tapes, several CD-ROMs). A configuration definition file is used to generate the table of contents for what is contained in the distribution.

In addition to being physical media, another useful way to think of a distribution is as a high-level division that can be used to distinguish between systems (e.g., GCSS, GCCS, ECPN), as shown in this example. The example would work equally well by defining the desired distributions one level lower in the tree and thus place responsibility for site configurations on a manager responsible for the site, rather than on a manager responsible for GCCS configurations at all sites. A distribution is not sufficiently detailed to permit the actual installation of any software since it must be decomposed further to the level of an actual workstation.

A *folder*, likewise, is a non-installable list of one or more folders, configurations, or bundles. Folders are used for organizational and display purposes only. A folder is not directly installable because it is organized at a level that spans multiple workstations and perhaps even multiple sites. In this example, the GCCS distribution media is composed of multiple folders at the top level representing geographically dispersed sites. The next lower level of folders is contained within a single site.

[6] The *distribution* term is a POSIX concept. It has been modified slightly in the *I&RTS* to include segments.

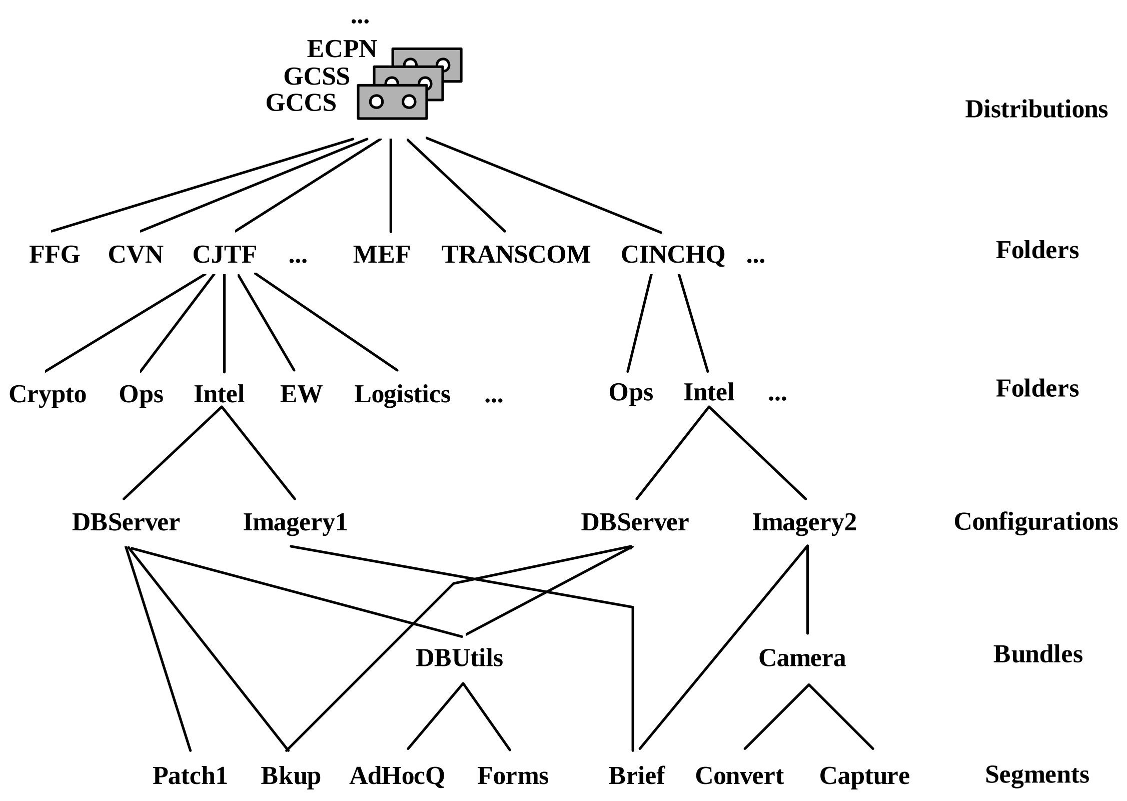

Figure 2-4: Configuration Definitions

A *configuration* is a list of bundles and/or segments that can be installed on a single machine. Configurations are mutually exclusive. That is, only one configuration can be installed on a single machine because there may be conflicts within the segments that comprise two different configurations. In the example, the `Intel` folder for the CJTF site contains two configurations for workstations: a database server configuration (`DBServer`) and an imagery configuration (`Imagery1`). A particular workstation may be loaded with a `DBServer` or an `Imagery1` configuration, but not both. This does not mean that imagery applications cannot reside on a database server. It only means that in this example, an engineering decision was made to prevent it from happening because of potential resource conflicts between the two configurations. If it were actually desirable to combine the applications in practice, a configuration could be defined which contained database server and imagery bundles. Or, desired segments could be selected individually for loading onto the platform.

A *bundle*[7] is a list of other installable bundles and/or segments. For brevity, Figure 2-4 does not show any bundles that contain other bundles. A bundle is directly installable, even if it contains further bundle definitions because the segments that comprise the bundle are checked when the bundle is created to verify that they do not conflict with one another. In

[7] The *bundle* concept is from POSIX, but has been modified slightly in the *I&RTS* to include segments.

the example shown, there are two bundles: `DBUtils` and `Camera`. The `DBUtils` bundle is used at both the `CJTF` and `CINCHQ` sites, but `Camera` is only used at `CINCHQ`.

There are several things to note about this example.

1. At installation time, the installer can use the installer tool to select a configuration and all of the appropriate segments will automatically be installed. The installer may also choose to decompose the configuration to look at individual bundles and segments and install them individually instead.

2. Configurations, bundles, and segments may be selected and installed directly without further selection on the part of the installer. Folders cannot; the installer must select some lower level in the hierarchy.

3. Folders may participate in multiple distributions or other folders; configurations may participate in multiple folders; bundles may participate in multiple configurations; and segments may participate in multiple bundles or configurations. Multiple participation is subject to the constraint that segments within a configuration or bundle cannot contain conflicting segments.

4. Configuration definitions are optional. They are provided as a convenience only. Also, it is possible to skip any of the levels in the configuration definition except for the lowest level (i.e., segments).

5. If a segment is selected twice either on purpose or as a result of how the configuration definition is constructed, it is actually installed only once.

6. Care should be taken in creating configuration definitions. It is wise to keep classified segments separate to avoid security management problems.

The same media can be used to load any workstation regardless of which site or in which space the workstation is located; however, during the installation process, only that portion of the configuration definition required for a particular workstation is actually loaded. The kernel COE is a required member of every distribution.

There are several advantages to configuration definitions:

- From a configuration management and security perspective, only one set of distribution media needs to be controlled. All software and data that are needed for the installation are contained on the media.

- From an installation perspective, the site installer only has one set of distribution media to worry about regardless of workstation use or hardware type. (The COE tools allow segments for multiple platforms to exist on the same physical distribution media. At installation time, the software determines the platform type and then makes available for selection only those segments that can execute on the platform.)

- From a system design perspective, the ability to create configuration definitions allows the flexibility of loading and executing only that software which is required to support a particular mission requirement.

2.1.4 DII Compliance

The degree to which “plug and play” is possible is highly dependent upon the degree to which segments are DII-compliant. DII compliance is defined to be an integer value that measures

- the degree to which a segment or system achieves conformance with the rules, standards, and specifications identified by the COE,
- the degree to which the segment or system is suitable for integration with the DII COE reference implementation, and
- the degree to which the segment or system makes use of COE services.

Appendix B contains a detailed checklist for areas where compliance is mandatory and an additional checklist for areas where compliance will be required in the future but are optional at present. The COE provides a suite of tools, described in Appendix C, which validate COE conformance.

By its very nature, an exhaustive list of “do's and don'ts” is not possible. DII compliance must be guided by overarching principles with checklists and tools to aid in detecting as many problem areas as possible. Full DII compliance embodies the following principles:

1. Segments shall comply with the guidelines, specifications, and standards defined in the *I&RTS*, the *Style Guide*, *DII Software Quality Compliance Plan*, and related documents such as the *JTA*.

2. Software and data shall be structured in segment format. Of necessity, COTS components of the bootstrap COE are exempted from this requirement. Segment format is described more fully in Chapter 5.

3. Segments shall be registered and submitted to the online library. The registration process is described in Appendix E while submission of segments to the online library is described in Chapter 10 and Appendix D.

4. Segments shall be validated with the `VerifySeg` tool prior to submission, and shall successfully pass the `VerifySeg` tool with no errors. An annotated listing of the `VerifySeg` tool output shall be submitted with each segment release.

5. Segments shall be loaded and tested in the COE environment prior to submission. Segment developers are responsible for testing their segment within the full COE kernel and with all COE-component segments that they depend upon. There is no requirement to include mission-application segments for which there is no dependency.

6. Segments shall fully specify dependencies, conflicts, and required resources through the appropriate segment descriptors defined in Chapter 5.

7. Segments shall be designed to be removable and tested to confirm that they can be successfully removed from the system. Some segments, especially COE components, are designed to be "permanent" but even these must be removable when a later segment release supersedes the current one.

8. Segments shall access COE components only through APIs published by DISA and segments shall not duplicate functionality contained within the COE. There is no requirement to integrate to COE functionality not required by the segment, but note that some segments may have an implied dependency on other segments.

9. Segments shall not modify the environment or any files it does not own except through environment extension files or through use of the installation tools provided by the COE.

The DII COE defines four areas in which compliance is measured, shown in Figure 2-5, called compliance *categories*. Within a specific category, a segment is assigned an integer value, called the compliance *level*, which is a measure of the degree to which a segment is compliant within that category. The DII COE takes this approach because it is especially useful in developing migration strategies for legacy systems. Compliance categories indicate the broad area in which a segment must be improved while compliance levels express the degree to which the segment meets COE objectives within that category.

The four DII compliance categories are:

Category 1: Runtime Environment. This category measures how well the proposed software fits within the COE executing environment, and the degree to which the software reuses COE components. It is an assessment of whether or not the software will "run" when loaded on a COE platform, and whether or not it will interfere with other segments. This category is closely related to, and is a way of measuring, interoperability.

Category 2: Style Guide. This category measures how well the proposed software operates from a "look and feel" perspective. It is an assessment of how consistent the overall system will appear to the end user. It is important that the resulting COE-based system appear seamless and consistent to minimize training and maintenance costs.

Category 3: Architectural Compatibility. This category measures how well the proposed software fits within the COE architecture (client/server architecture, DCE infrastructure, CDE desktop, etc.). It is an assessment of the software's potential longevity as the COE evolves. It does *not* imply that all software must be based on client/server or RPC (Remote Procedure Call) techniques. It simply means that a reasonable design choice has been made given that the specific architectural characteristics of the COE reference implementation.

Category 4: Software Quality. This category measures traditional software metrics (lines of code, McCabe complexity metric, etc.). It is an assessment of program risk and software maturity.

Runtime Environment	**Style Guide**	**Architectural Compatibility**	**Software Quality**
0 $\longrightarrow$ **j**	**0** $\longrightarrow$ **k**	**0** $\longrightarrow$ **n**	**0** $\longrightarrow$ **m**

Figure 2-5: DII Compliance Categories and Levels

Note: While there are four compliance categories, style-related items are included within the *I&RTS* checklist. Specifications within the *Style Guide* are mapped to these items at the appropriate compliance level where they are included. For example, Category 1 (Runtime Environment) Level 5 compliance requires adherence to the “look and feel” of the native GUI. The *Style Guide* contains a checklist for verifying that a segment conforms to the native GUI.

These four categories attempt to quantitatively answer the following questions about a proposed addition to the system:

- (Category 1: Runtime Environment) Can the proposed software be added to the system? Will it adversely affect system interoperability?
- (Category 2: Style Guide) Is the proposed software user-friendly? Will it make the system appear seamless to an end user?
- (Category 3: Architectural Compatibility) Is the proposed software architecturally sound and in line with where the COE is going? Will technology advances quickly obsolete the proposed software?
- (Category 4: Software Quality) What is the program risk? Will significant program expenditures be required for life-cycle maintenance of the product?

The principles and techniques described in the remainder of this subsection apply to each of the compliance categories. However, only the compliance levels for the Runtime Environment Category will be discussed any further.

The COE defines eight progressively deeper levels of integration for the Runtime Environment Category. These levels are directly tied to the degree of interoperability achieved as is described in subsection 2.1.5. Note that levels 1-3 are “interfacing” with the COE, not true integration. True integration begins at level 4.

Level 1: Standards Compliance Level. A superficial level in which the proposed capabilities share only a common set of COTS standards. Sharing of data is undisciplined and minimal software reuse exists beyond the COTS. Level 1 may, but is not guaranteed to, allow simultaneous execution of the two systems.

Level 2: Network Compliance Level. Two capabilities coexist on the same LAN but on different CPUs. Limited data sharing is possible. If common user interface standards are used, applications on the LAN may have a common appearance to the user.

Level 3: Workstation Compliance Level. Environmental conflicts have been resolved so that two applications may reside on the same LAN, share data, and coexist on the same workstation as COE-based software. The kernel COE, or its equivalent, must reside on the workstation. Segmenting may not have been performed, but. some COE components may be reused. Applications do not use the COE services and are not necessarily interoperable.

Level 4: Bootstrap Compliance Level. All applications are in segment format and share the bootstrap COE. Segment formatting allows automatic checking for certain types of application conflicts. Use of COE services is not achieved and users may require separate login accounts to switch between applications.

Level 5: Minimal DII Compliance Level. All segments share the same kernel COE and functionality is available via the Executive Manager. Boot, background, session, and local processes are specified through the appropriate segment descriptor files. (See Chapter 5 for a description of the types of processes.) Segments adhere to the basic “look and feel” of the native GUI, as defined in the *Style Guide*. Segments are registered and available through the online library. Applications appear integrated to the user, but there may be duplication of functionality and full interoperability is not guaranteed. Segments may be successfully installed and removed through the COE installation tools. Database segments are identified as unique or sharable according to their potential for sharing.

Level 6: Intermediate DII Compliance Level. Segments utilize existing account groups, and reuse one or more COE-component segments. Minor documented differences may exist between the *Style Guide* and the segment's GUI implementation. Use of non-standard SQL in database segments is documented and, where applicable, packaged in a separate database segment.

Level 7: Interoperable Compliance Level. Segments reuse COE-component segments to ensure interoperability. These include COE-provided communications interfaces, message parsers, database segments, track data elements, and logistics services. All access is through published APIs with documented use of few, if any, private APIs. Segments do not duplicate any functionality contained in COE-component segments. The data objects contained within a database segment are standardized according to DOD 8320 guidance.

Level 8: Full DII Compliance Level. Proposed new functionality is completely integrated into the system (e.g., makes maximum possible use of COE services) and is available via the Executive Manager. The segment is fully compliant with the *Style Guide* and uses only published public APIs. The segment does not duplicate any functionality contained elsewhere in the system whether as part of the COE or as part of another mission application or database segment.

Bootstrap Compliance (Level 4) is required before a segment may be submitted to DISA for evaluation as a prototype. Such segments will not be fielded nor accepted into the online library. At DISA's discretion, segments which meet the criteria for Minimal DII Compliance (Level 5) may be accepted into the online library, and installed at selected sites as prototypes for user evaluation and feedback. Such segments will not be accepted as fieldable products. Acceptance as an official DISA fieldable product requires demonstration of Interoperable Compliance (Level 7) and a migration strategy to Full DII Compliance (Level 8), unless the proposed segment is an interim product that is targeted to be phased out in the near term.

The compliance categories and levels defined here are a natural outcome of developing a reasonable approach to migrating legacy systems into the COE. The first step of Category 1, covered by Levels 1-4, is to ensure that systems do not destructively interfere with each other when located at the same operational site. Level 5 is sometimes called a “federation of systems” in that systems are still maintained as “stovepipes,” but they can safely share common hardware platform resources. Levels 6-8 complete the approach by reducing functional duplication, promoting true data sharing, and making the system appear to the user as if it were developed as a single system. The last three levels represent varying degrees of integration from marginally acceptable (Level 6) to a truly integrated system (Level 8). All 8 levels represent a progressively deeper level of interoperability.

The same compliance levels apply to SHADE databases, as well. The majority of the SHADE issues in Levels 1-4 are concerned with proper use of the COTS database management systems’ functionality and with not destructively accessing data belonging to other databases. At Level 5, a database must identify those components of its schema which are candidates for “sharing.” Levels 6-8 reduce and then eliminate data sets that are redundant with information in shared and universal segments, including database design modifications and data migration and cleansing to provide interoperability in both data structure and content.

Compliance checking is done on a segment-by-segment basis according to the definitions given here and through the checklist approach in Appendix B. The categories and levels described here are independent of where the segment fits into the system. That is, the same definitions apply whether the segment is a COE-component segment or a mission-application segment. However, it is sometimes necessary to compute the compliance level of a collection of segments. This is called a *composite compliance level*. The remaining subsections below describe how to compute a composite compliance value for an arbitrary group of segments, for the COE itself, for a COE-based system, and for systems which contain both COE and non-COE based computing platforms. A composite value is required because otherwise a system is only as compliant as its least compliant segment and the least compliant segment may be in the COE[8] itself. Thus, the intent is to not penalize systems for non-compliant components in the COE itself.

Strictly speaking, discussion of DII compliance requires qualification with a category name, a compliance level, and whether compliance is being measured against a segment or a collection of segments. Thus, it is correct to say that a particular segment is Category 1, Level 4 compliant, but it could be confusing to omit the qualifier Category 1. Because of widespread usage in the COE community, when a category is not stated, Categories 1 and 2 are assumed.[9]

The *I&RTS* expressly uses integer values rather than decimals or percentages to state DII compliance. Expressing compliance as a percentage is both confusing and misleading. For example, to state that a segment is 85% Level 6 compliant can be interpreted in many ways. It could mean that 85% of the effort required to achieve Level 6 compliance has been achieved, or that 85% of the functionality in the system is 85% Level 6 compliant. However, it most likely means only that the segment successfully passes 85% of the Level 6 criteria in Appendix B. Because of the difficulty in precisely interpreting the intended meaning, only integer compliance values are allowed. Otherwise, it is difficult to quantitatively compare two segments or systems if both claim to be 85% Level 6 compliant.

2.1.4.1 Compliance for an Arbitrary Group of Segments

Segments are often grouped together, as in a configuration definition. The composite compliance level for an arbitrary collection of segments is the compliance level for the *least* compliant segment.[10] For example, suppose a group of four segments have compliance levels of 5, 8, 3, and 8 respectively. Then the composite compliance level for this group of four segments is 3.

[8] The COE reference implementation contains software contributed by legacy systems. It may not be cost effective to expend the effort to achieve full Level 8 compliance for some of these legacy contributions because they are going to eventually be phased out. In the interim, systems that use these segments should not be penalized for their lack of compliance.

[9] The *JTA* states a requirement for a minimum of Level 5 compliance as does OSD directive. In both cases, Category 1: Runtime Environment and Category 2: Style Guide are intended. The requirement is levied on individual segments, and on COE-based systems.

[10] This is how the compliance level for an aggregate segment is measured. (See Chapter 5 for the definition of an aggregate segment.) The compliance level of an aggregate segment is the compliance level of the *least* compliant segment in the aggregate.

This approach to calculating composite compliance levels intentionally places a heavy penalty on groups that have segments with low compliance levels and gives no “credit” if there are segments with high compliance levels. An alternative approach would be to average the levels, but because compliance is a direct measure of interoperability and because artificially increasing the number of segments could have the misleading effect of boosting the apparent level of compliance, this approach was rejected.

2.1.4.2 Compliance for the DII COE

Calculating the compliance level for the COE itself requires computing the composite compliance level for 1) the COE kernel, and 2) for the Infrastructure Services and Common Support Applications layers. As described in subsection 2.1.4.1, the composite compliance level for each of these two groups of segments is the level of the least compliant segment in the group.

Let C_k be the composite compliance level of the COE kernel. Let C_c be the composite compliance level for the combined Infrastructure Services and Common Support Applications segments. Then the composite compliance level for the DII COE (C_{dii}) is given by the equation

$$C_{dii} = \mathrm{TRUNC}([C_k + C_c]/2)$$

where **TRUNC** means to truncate the result to an integer value.

Consider an example. Assume the kernel has three segments with compliance levels 6, 8, and 5. Assume there are four segments in the Infrastructure Services layer with compliance levels 8, 8, 7, and 4. Lastly, assume that there are seven segments in the Common Support Applications layer and all are level 8 compliant.

The composite compliance level for the combined Infrastructure Services and Common Support Applications segments is the compliance level of the least compliant segment (e.g., 4). Thus, the following gives the composite compliance level for the DII COE for this example:

$$C_k = 5$$
$$C_c = 4$$
$$C_{dii} = \mathrm{TRUNC}([5 + 4]/2) = \mathrm{TRUNC}[9/2] = 4.$$

2.1.4.3 Compliance for a COE-Based System

The composite compliance for a COE-based system is computed in a manner similar to that of computing the compliance for the DII COE. The approach is to compute the composite compliance level for the mission-application segments and then factor in the DII compliance. The computation here is valid only if every platform in the system is COE-based. If there is a mixture, refer to subsection 2.1.4.4.

Let C_{ma} be the composite compliance level of all the mission applications in the system. Let C_{dii} be the composite compliance level for the COE computed as described in subsection 2.1.4.2. In computing C_{dii}, only those segments in the COE that are actually used in the resulting system are considered. Then the system COE composite compliance level, C_s, is computed as follows:

If $C_{ma} < C_{dii}$, then $C_s = \mathrm{TRUNC}[(C_{ma} + C_{dii})/2]$,
else $C_s = \mathrm{ROUND}[(C_{ma} + C_{dii})/2]$

where **ROUND** means to round the result to the nearest integer.

As an example, assume that a system has five mission applications with compliance levels of 5, 7, 7, 8, and 8. Assume that the DII compliance level for the COE segments actually used in the system is 6. Then the system composite compliance level is

$$C_{ma} = 5$$
$$C_{dii} = 6$$
$$C_s = \mathrm{TRUNC}[(5 + 6)/2] = 5.$$

If the least compliant segment (level 5) could be improved to reach level 7 with no change in the COE, then the resulting system compliance level would be increased to 7.

2.1.4.4 Compliance in Mixed Systems

COE-based systems are likely to be created which include a mixture of COE-based and non-COE based computing platforms. This may occur for several reasons:

1. because required functions in the target system have not yet migrated to the COE,
2. because of the need to interface with legacy systems that are not COE-based (e.g., mainframe applications),
3. because the COE is not presently available on a required platform, or
4. because the platform is highly specialized and is not appropriate for the COE.

An example of the latter situation is a receiver subsystem that contains dedicated hardware for direct receiver control. A system built around such components is likely to use a platform on which the COE is available for operator interaction and for receiver tasking, and hence would be a mixed system.

Calculating the system composite compliance for all four situations is done just as with COE-based systems described in subsection 2.1.4.3. In the first situation above, the application that contains the required functionality can still be evaluated against the compliance checklist and so arrive at a compliance level. The resulting system compliance level will likely be very low.

Computation of the system composite compliance in the last three situations is equally straightforward. Compliance is computed by ignoring the legacy platforms and platforms for which the COE is not available.

2.1.5 Interoperability of COE-Based Systems

This subsection describes interoperability in the context of the COE, and shows the relationship between DII compliance levels and interoperability. But first, it is important to distinguish between *interfacing*, *integration*, and *interoperability*. The three terms are closely related and often confused, but they are distinct concepts. Proper understanding of the interrelationship of these three terms makes it clear that the DII COE is an approach towards integration that goes beyond simple interfacing or “peaceful coexistence” to true interoperability.

2.1.5.1 Interfacing Systems

Interfacing is the ability of two systems to exchange data, typically by converting data to an agreed-upon intermediate format. Interfacing should be viewed as one approach towards achieving interoperability, or as a first-level approximation of interoperability. For example, military systems frequently interface with one another by exchange of USMTF messages. They are able to “interoperate” to the extent that they can pass meaningful data to one another in an agreed-upon intermediate USMTF format.

Interfacing provides a limited degree of interoperability but fails to fully satisfy “real-world” operational requirements. Interfacing

- requires consistent interpretation of the agreed-upon format for data exchange;
- requires systems to stay in synch as the data exchange format changes;
- may result in loss of precision or other attributes (e.g., one system may process latitude and longitude only in degrees and minutes, while another system may process latitude and longitude down to decimal fractions of a second); and
- fails to ensure that applications interpret the exchanged data consistently.

For these reasons, successful “interfacing” is often limited to a specific version of the two systems in question and may not survive when an upgrade to either system is performed. Also note that standards profiles specify how interfacing can be accomplished.

2.1.5.2 System and Segment Integration

Integration is often used to refer to integration **within** a system or **between** systems, or to refer to software and data integration. Within the context of this document, integration refers to combining segments to create a system. Segment integration refers to the process of ensuring that segments:

- work correctly within the COE runtime environment;
- do not adversely affect one another;
- conform to the standards and specifications described in this document;
- have been validated by the COE tools; and
- can be installed on top of the COE by the COE installation tools.

Integration requires resolution of compatibility issues between components that are to be interconnected. Integration attempts to allow sharing of a common resource (such as data) without the need for intermediate translations from one format to another. Note that the COE is a technique for achieving both software and data integration; it is the SHADE component of the COE which the technique for assuring data integration. But the DII COE goes further because COE/SHADE-type integration for software and data provides true interoperability as a byproduct. The COE with full SHADE does not create any technical roadblocks to interfacing, but does strongly encourage a deeper level of integration that promotes true interoperability.

Integration of a segment with the COE is the responsibility of the segment developer. Government integrators perform integration of the system as a whole and interoperability testing.

2.1.5.3 Interoperability Levels

In the context of this document, *interoperability* refers to the ability of two systems to exchange data:

- with no loss of precision or other attributes,
- in an unambiguous manner,
- in a format understood by and native to both systems, and
- in such a way that interpretation of the data is precisely the same.

There are two significant differences between interoperability and interfacing. The first is that with interoperability the exchange of data is performed without the need to translate to an intermediate format, such as a USMTF message format. This leads to the second difference in that interoperable systems will produce exactly the same “answer” in the presence of identical data. Systems that are interfaced will not necessarily do so because of the potential loss of precision or data in the data exchange.

The concept of interoperability is explored in more detail in a study sponsored by the C4ISR Integration Task Force Integrated Architectures Panel. The draft document proposes four levels[11] of interoperability which are adopted by the *I&RTS*. The four proposed levels are as follows, listed in decreasing order of interoperability:

> **Level A: Universal - Virtual C4I System.** This level represents the ultimate goal of full interoperability. Universal interoperability is characterized by the ability to globally share integrated information in a distributed information space. Another way to view Universal interoperability is as a way to globally share systems.
>
> **Level B: Advanced - Integrated Systems.** The Advanced level of interoperability is characterized by shared data between applications, including shared data

[11] The draft document also proposes a mapping between the *I&RTS* compliance levels and interoperability levels. However, the mapping fails to properly account for integration when *identical* software is used for common functions.

displays, and information exchange through a common data model. This level provides for sharing of information in a distributed but localized environment and for sharing of applications.

Level C: Intermediate - Distributed Systems. This level is characterized by a client/server environment with standardized interfaces and distributed computing services that allow for exchange of heterogeneous data (e.g., maps with overlays, annotated images), and advanced collaboration. This level of interoperability is achievable with implementation of "cut and paste" between applications, through World-Wide-Web technology, and through basic use of DII COE features.

Level D: Basic - Discrete Systems Interaction. A primitive level of interoperability characterized by peer-to-peer connected systems that allows basic exchange of homogenous data (e.g., email, formatted messages) and allows for basic collaboration. This level of interoperability is achievable by interfacing techniques described above and by use of standard office automation products that provide data import/export functions for handling data from another product.

2.1.5.4 Mapping Interoperability and Compliance Levels

Note that progressing from one level of interoperability to a higher one requires a deeper degree of integration, more commonality in the infrastructure building blocks, and a greater ability to share data and information. These are precisely the requirements for progressing to deeper levels of DII compliance, and can be achieved through the use of COE/SHADE facilities. Moreover, two systems which are completely identical achieve the highest possible degree of interoperability, so the more software reuse is achieved, the greater the degree of interoperability. Thus, there is a direct relationship between integration, reuse, DII compliance levels, and interoperability.

Integration alone does not imply interoperability; it only provides a level of assurance that the system will work as designed. However, when COE-based systems are integrated together, interoperability is achieved as a byproduct because common software is used for common functions. The degree to which interoperability is achievable is dependent upon the degree to which the two systems are DII-compliant. Universal Interoperability can only be achieved when systems use exactly the same software to perform identical functions and use the same database segments for required data elements. Implementation of agreed-upon paper standards is not itself sufficient.

Table 2-1 shows a mapping between DII compliance levels and interoperability levels. The transition and correspondence between levels is not sharp, as the table suggests because the purpose and focus are different for the two different types of levels.

DII Compliance Levels	Interoperability Levels
1. Standards	Basic
2. Network	Basic
3. Workstation	Basic, Intermediate
4. Bootstrap	Basic, Intermediate
5. Minimal	Basic, Intermediate
6. Intermediate	Intermediate
7. Interoperable	Intermediate, Advanced, Universal
8. Full	Advanced, Universal

Table 2-1: Compliance and Interoperability Levels

2.1.6 Principles for Selecting COE Components

Selection of the specific software modules that comprise the COE determine which mission domain(s) can be addressed by a particular COE reference implementation. But selection of COE components is not arbitrary: it is driven by a number of important architectural and programmatic principles. First, there is a determination of what functions the COE is required to perform, then there is a set of criteria for selecting software components which perform the required functions. A function is part of the COE if it meets one or more of the following criteria:

1. *The function is part of the minimum software required to establish an operating environment context.* This is normally provided by COTS products and includes the operating system, windowing software, security software, and networking software.

2. *The function is required to establish basic data flow through the system.* To be useful, a system must have a means for communicating with the external world. To be efficient, consistent, and robust, a system must also have standard techniques for managing data flow internal to the system.

3. *The function is required to ensure interoperability.* Standards alone cannot guarantee interoperability, but using common software for common functions and using shared and universal database segments with DOD 8320 standard data objects comes much closer. As an example from the GCCS mission domain, an OTH-GOLD message parser is part of the COE because interoperability cannot be achieved if two different message parsers implement a different set of assumptions about the OTH-GOLD message specification or use a different specification revision.

4. *The function is of such general utility that if rewritten it constitutes appreciable duplicative effort.* This includes printer services, an alerts service for disseminating alerts, and a desktop environment for launching operator-initiated processes.

Subsections 2.1.1 and 2.1.2 detail the functions currently defined to be in the DII COE. The first three criteria listed above are technical in nature because they dictate from an architectural perspective what software *must* be contained in the COE for a given mission domain. The fourth criteria, however, is more programmatic in nature because it is often a tradeoff between the cost of modifying a legacy system to remove duplication versus the cost of maintaining duplicative code, the cost of potentially requiring additional hardware resources because of duplication, and the cost of operator training when there are different ways to accomplish the same action. DII compliance requires that there be no duplication of functions in the first three criteria but some flexibility is possible for the fourth.

There are two frequently voiced concerns about COE services:

1. if a module becomes part of the COE, it cannot be easily changed or customized; and
2. the larger the COE is, the more inflexible and the poorer the performance of the resulting system.

The first statement is partially true and is so by design. It is essential to perform careful configuration management of COE components, and they must be changed only in a controlled way in response to formally reported problems. Stability of the COE is crucial to the system, so modifications must be done carefully, deliberately, and at a slower pace than changes in non-COE routines. But just because changes are controlled does not mean that the COE routines cannot be customized. Ongoing work in the COE is to devise and refine techniques to “open up the architecture” to allow applications to customize COE components in ways that do not violate COE principles and do not adversely impact other developers using COE services.

The second statement is a misunderstanding of the COE architecture and concept. Unlike many systems, the COE is not designed as a single monolithic process, but is instead designed as a collection of relatively small processes. While a small number of these are loaded into memory as background processes, most are loaded into memory on demand in response to operator actions (e.g., edit a file, display a parts inventory) and only for the amount of time required for them to perform their task. This approach offers considerable flexibility because it limits the number of background processes required. Except for cases where segments require adding new background processes, adding new segments does not adversely impact performance. The price paid is a small amount of overhead required to load functions on demand, but this is generally negligible because the overhead is small and comes usually in response to an operator request to bring up a display that must respond only at human speeds.

A COE-component segment is not necessarily installed on every target workstation or as part of every COE-based system. A COE-component segment can be omitted from the system or installation if:

- *Any remaining COE-component segments do not require the functionality provided by the segment.* For example, the COE provides a number of message parsers for processing military message formats. But systems such as ECPN have no need to handle military message formats and therefore such parsers need not be included in the ECPN system. However, in many cases there is no real advantage to deleting a COE-component segment because it will not be activated unless required and the amount of disk space taken up is small. Eliminating the function will increase the burden of configuration management problems more than leaving the function in the system.

- *The functionality provided by the segment is not required by any remaining COE-component segments.* Selection of certain functions within the COE automatically dictates the inclusion of segments on which those functions depend. This is not the same as saying that the COE is not modular. On the contrary, it is an observation that inclusion of a higher-level function requires inclusion of all lower-level routines used to build the function. This is a direct consequence of modularity, not a contravention.

- *The functionality provided by the COE segment is not duplicated by another segment.* A common pitfall to avoid is omitting a COE component because its functionality is available through some other means. The problem with this approach is that a common "look and feel" and consistent operation are no longer preserved between applications, and interoperability may be reduced.

Omission of COE-component segments that are not required is done automatically by the COE installation software.

2.2 Account Groups and Profiles

In a typical operating system, users are assigned individual login accounts. Configuration files are created to establish user preferences and a runtime environment context. In the Unix operating system, configuration files (for example, `.cshrc`) establish the runtime environment context for the user. COTS products such as CDE also have configuration files that contribute to the runtime environment context as well. These configuration files must be set up and established for each user of the system. The COE provides standard versions of the required “dot” files. These should be used when creating account groups because they standardize the operation of the system across all account groups, and because the COE-provided files demonstrate how to support dynamic profile switching.

An *account group segment* is a template used within the COE for setting up individual login accounts and a required runtime environment context. Account groups contain template files for specifying COE processes to launch at login time, functions to be made available to operators, and global default preferences such as color selections for window borders.[12] Account groups are described further in Chapter 5 of this document.

Account groups can also be used to perform a first-level division of operators according to how they will use the system. This technique is used in the COE to identify at least five distinct account groups:

- Privileged Operator (e.g., root) Accounts,
- System Administrator Accounts,
- Security Administrator Accounts,
- Database Administrator Accounts, and
- Non-Privileged Operator Accounts.

Other account groups may exist for specialized system requirements, such as providing a character-based interface, but all account groups follow the same rules. Within an account group, subsets of the available functionality can be created. These subsets are called *profiles*. An operator may participate in multiple account groups with multiple profiles, and can switch from one profile to another without the need to log out and log in again. An operator may also select multiple actives profiles to provide an operational environment from a collection of account groups. For example, assuming the operator has appropriate permissions, an operator may combine a profile based on the System Administrator account group with a profile based on a Database Administrator account group.

Figure 2-6 shows the hierarchical relationship among account groups, profiles, and individual users. It is intended to convey several points.

- Multiple profiles may be assigned to an account group, but a particular profile may be assigned to only one account group. Assuming the operator has proper permissions,

[12] The user may modify preferences, but the Account Group establishes the initial, default settings.

multiple profiles may be selected at one time to give the operator features from multiple account groups at the same time.

- Multiple operators may be assigned to the same profile. For example, operator Op4 and operator Opn are shown assigned to the same profile within the Non-Privileged Operator account group.

- Operators may be assigned to multiple profiles either within the same account group, or across account groups. Opn is assigned to three profiles within the Non-Privileged Operator account group. Op3 is assigned to a profile in the System Admin, Security Admin, and Database Admin account groups.

- Not only can an operator be assigned to multiple profiles, but multiple profiles may be active at a time. The operator may switch between profiles without the need to log in and out. (Optionally, a system can be configured to permit a single profile at one time.)

- The COE allows profiles to be locked. That is, if Op4 and Opn are assigned to the same set of profiles, the system can be configured so that if Op4 is in a specific profile first, then Opn is locked out from using that profile until Op4 is no longer using it.

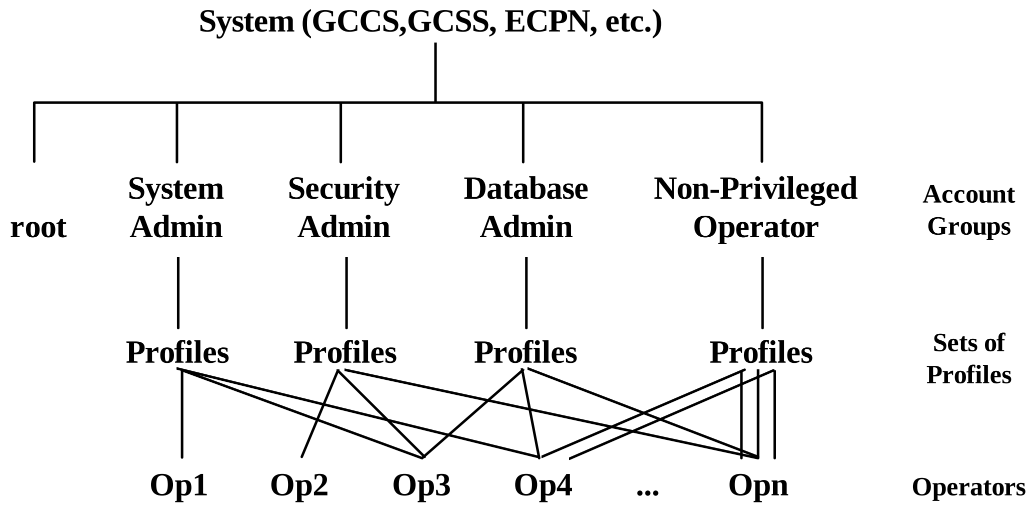

Figure 2-6: Account Groups, Profiles, and Users

2.2.1 Privileged User Accounts

Most operating systems provide a privileged "super user" account. Both Unix and Windows NT have the concept of a privileged account. Privileged accounts are normally restricted to knowledgeable systems administrators because serious damage can be done to the system if they are used improperly. Security requirements also dictate careful control and auditing of actions performed when operating as a privileged user.

The COE design philosophy is to not require the use of a privileged user account for normal operator activities. Certain processes (such as setting the system time in Unix) cannot be performed without superuser privileges, but such privileges are given to the process, not the user, and only for the period of time necessary to perform the required action. Root-level access need *not* be provided to the user for such actions: indeed, it should not be provided.

Normal operation does not require a command-line-level access, especially to root. Command-line access for any COE segment is expressly prohibited unless the DISA DII COE Chief Engineer grants prior approval. However, a privileged user account is preserved in the system for use by trusted processes, for unusual system administration tasks or installations, and for abnormal situations where “all else fails.”

2.2.2 Security Administrator Accounts

Security in the COE is implemented through a collection of security services and trusted applications. One such trusted application is the Security Administrator application that allows a Security Administrator to monitor and manage security. Precise functionality of the Security Services provided is vendor-dependent because vendors have taken different approaches. Security features in Windows NT and Unix are significantly different, but even within Unix, security features vary considerably from one vendor to another.

The Security Services are loaded as part of the bootstrap COE and COE kernel. (The precise sequence for loading security software is vendor-dependent.) The Security Administrator application is designed to be made available to only a restricted group of operators. Available functions include the following:

- Ability to create individual login accounts
- Ability to create defined operator profiles, including granting database privileges as established by the Database Administrator (DBA)
- Ability to create/modify database user accounts
- Ability to assign read, write, and modify data permissions
- Ability to customize menus by operator profile
- Ability to export accounts and profiles to other LAN workstations
- Ability to review and archive the audit log.

2.2.3 System Administrator Accounts

The System Administrator Account Group is a specialized collection of functions that allow an operator to perform routine maintenance operations. This software is designed to be made available to a restricted group of operators. It is loaded as part of the COE kernel (some features may be part of the bootstrap COE, depending upon the vendor) because it contains the software required to load segments. Functionality provided includes:

- Ability to format floppy disks
- Ability to install and to remove segments
- Ability to set workstation name and IP address
- Ability to install and configure printers
- Ability to create and to restore backup tapes
- Ability to shutdown and to reboot the system
- Ability to configure network host tables
- Ability to configure and manage the network.

2.2.4 Database Administrator Accounts

The Database Administrator Account Group is to be used by those individuals responsible for performing routine database maintenance activities such as backups, archives, and reloads. The specific capabilities are dependent upon which commercial relational database software is in use and upon tools provided with these commercial products.

Functions included within this account group are:

- Ability to archive and restore database tables
- Ability to import and export database entries
- Ability to create/modify database user accounts
- Ability to checkpoint and journal database transactions.

Note: User account management is normally done as part of a Security Administrator account. However, the COE provides the ability to modify the database portion of already created user accounts via the Database Administrator accounts as well. Only those user account items related to database administration can be modified by the database administrator.

2.2.5 Operator Accounts

Most operators will not require, nor will site administrators grant access to, capabilities described in the previous subsections. Most system users will be performing mission-specific tasks such as creating and disseminating Air Tasking Orders (ATOs), preparing briefing slides, performing ad hoc queries of the database, participating in collaborative planning, etc. The precise features available depend upon which mission-application segments have been loaded and the profile assigned to the operator.

2.2.6 Character-Based Interface Accounts

Certain legacy systems require the ability to provide a character-based interface to the user. This is typically required for remote users where the communications bandwidth is too low to support a GUI-based application or because the user's hardware does not support graphics (e.g., VT100 terminals).

The COE provides a character-based account group for such situations. These may have profiles defined just as with any other account group. When the user logs in, a menu of options, such as

0) Exit
1) AdHoc Query
2) TPFDD Edit

Enter Option:

is presented to the user.

Character-based account groups are restricted in the sense that a user account is either character-based or it is not. If an operator has access to a terminal that supports a GUI interface and to another that does not, the operator must have two separate login accounts: one which uses character-based profiles and one which does not.

2.3 Site Configurations

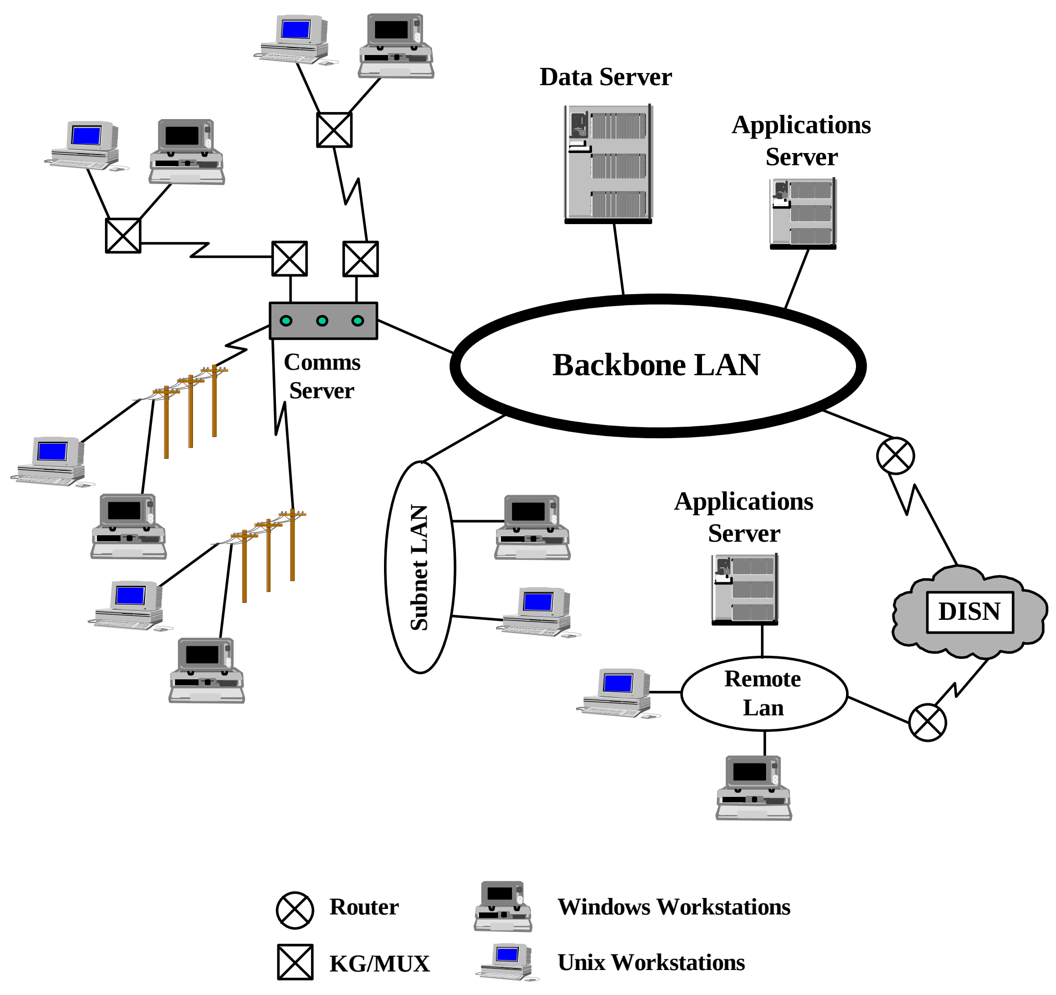

Figure 2-7: DII Notional LAN Architecture

Figure 2-7 is a simplified notional LAN diagram for a typical COE-based system. The architecture consists of a 3-tier client/server environment incorporating data servers, application servers, and workstations interconnected on a LAN/WAN. The division shown separates data (data servers), functions (application servers), and presentation (workstation). System components are interconnected on a LAN/WAN through direct connection to a LAN, through subnets connected over routers, through dedicated lines, or via dial-up through a communications server. Cryptologic equipment may be installed to secure communications over non-secure lines as shown. Remotes with limited bandwidth will not generally have access to the complete suite of mission applications available to local users.

In a typical installation, there will be one or more database servers and several application servers. The database server is the repository for all databases and may be replicated at strategic places in the LAN architecture to improve performance and to balance loading. COE services ensure that replicated databases stay synchronized.

A typical installation will often include other servers that are not shown in the diagram. A Web server connected to the outside world through a firewall allows sites to take advantage of Web technologies for collaborative planning purposes. As described in a later chapter, the COE also provides facilities that allow developers to create applications that use the Web for accessing applications and data.

Network management is greatly simplified if a domain name server is created and if there is a server for management of user accounts and profiles. Segment installation is simplified by designating servers that allow workstations to be loaded across the LAN rather than individually from magnetic media. This approach also simplifies software distribution because when software updates are received, they can be tested in isolation, then loaded onto a segment server for distribution to affected workstations. Combined with configuration definitions, the COE provides powerful tools for managing software installation and distribution.

The COE supports both “network-centric” and “workstation-centric” LAN management. Network-centric refers to the ability to centrally manage network resources (e.g., user accounts, profiles, software installation). In keeping with COE principles, centralized management can be done from any workstation (subject to security considerations) with infrastructure services responsible for effecting the changes across the network. Workstation-centric refers to the ability to distribute network management. The choice of centralized versus distributed is a preference which may vary across distributions or sites.

2.4 Installing COE-Based Systems

Figure 2-8 is a notional depiction of the installation process. (It should not be interpreted too literally since vendor-specific loading instructions may require slight alterations in the loading sequence shown.) First, the operating system, windowing environment, and any necessary patches are loaded as per vendor instructions. Then, the COE Security Administration software and System Administration software are copied onto disk with the equivalent of a Unix "tar" command. The segment installation tool is copied onto disk as part of installing the System Administration software and installation of the System Administration software is done in such a way as to also create a System Administrator operator account. This completes installation of the bootstrap COE. Next, the operator logs in as a system administrator, invokes the segment installation tool, and selects the remaining kernel COE segments for installation. Finally, any remaining mission-specific segments are selected and loaded.

This installation approach has several advantages. It greatly simplifies the installation process by handling all vendor-unique issues first (e.g., loading the operating system and patches). It guarantees a standard, known starting configuration for all platforms regardless of how they will be used. It allows all remaining segments to be loaded in a standard way regardless of the hardware platform or mission application, thus simplifying system administration. Through the COE, it allows segments to extend the base environment as required as they are loaded.

Figure 2-8 describes the general flow for installing a system, which can be accomplished in either a "pull" or a "push"[13] mode. In pull mode, installation is done locally from the target platform. In push mode, installation is performed remotely onto the target platform from a different platform.

Installation may be accomplished in several ways:

- directly from distribution media (e.g., tape, CD-ROM),
- locally from distribution media mounted on a different platform,
- across the network from a segment server, or
- through a Web browser interface from a centralized segment server.

The distribution media and servers may contain executables for multiple hardware platforms. The segment installation tool ensures that only those executables which are compatible with the target platform (e.g., NT, Solaris, DEC) are selectable and hence installable.

[13] Installation in a "push" mode requires that the COE kernel already be installed on the target machine and that the target machine already be accessible from the network.

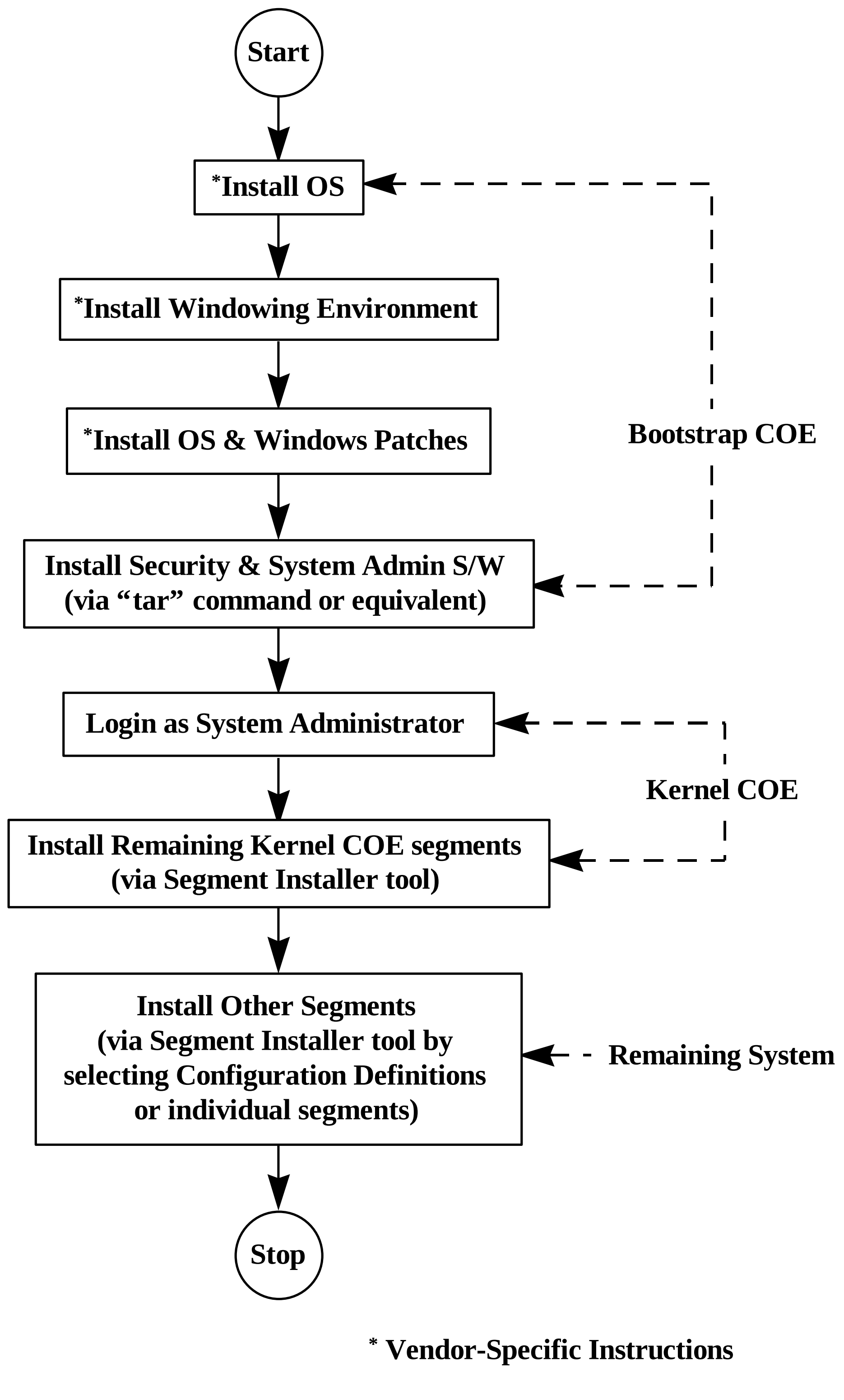
Start
*Install OS
*Install Windowing Environment
*Install OS & Windows Patches
Install Security & System Admin S/W (via "tar" command or equivalent)
Login as System Administrator
Install Remaining Kernel COE segments (via Segment Installer tool)
Install Other Segments (via Segment Installer tool by selecting Configuration Definitions or individual segments)
Stop
Bootstrap COE
Kernel COE
Remaining System
* Vendor-Specific Instructions

Figure 2-8: Installing a COE-Based System

2.5 Supported Configurations

The DII COE is an open architecture and as such is not tied to a specific hardware platform. It uses POSIX-compliant operating systems and industry standards such as X Windows, Motif, and NT. In actual practice, POSIX compliance and industry standards have not progressed to the point where verification that software works in one hardware/software configuration is a guarantee that it will work in another. COTS vendors do not necessarily provide backwards compatibility with subsequent releases, and in fact much of the effort consumed in porting the COE from one configuration to another is to account for lack of compatibility between vendors or between vendor releases. Thus, what hardware/software configurations to support is more an issue of testing and life-cycle maintenance than it is one of "openness" or software portability.

COE reference implementations exist for a number of platforms. The list of supported hardware and software components is growing as the COE and COE-based systems evolve to meet operational requirements. Appendix A lists the current DISA-supported COE configurations. It also describes a DISA "self-certification" program that allows vendors or services to receive copies of the COE kernel in order to port it to platforms or operating systems not currently supported by DISA. DISA will test the ported kernel to ensure it meets COE requirements and will issue a certification for the specific platform. Responsibility for supporting the ported COE on the new platform is the responsibility of the vendor/service that has funded the effort.

Appendix A will be updated as required to reflect new hardware/software configurations. Note that not all of the COTS products listed in Appendix A are part of the COE kernel and thus are not required for every workstation. Refer to the DISA DII COE Chief Engineer for requirements for other platform or COTS software versions, or for an updated list of supported vendor products.

Specifying precise hardware requirements in terms of memory, disk space, etc. is a function of whether the workstation is a shared data server, an application server, or a client workstation, and whether the workstation is standalone or on a network with other COE-based workstations. Consult the DISA DII COE Chief Engineer for hardware configuration options.